Spatial and Epistemic Modalities in Constraint-Based Process Calculi

Sophia Knight, Catuscia Palamidessi, Prakash Panangaden, Frank D. Valencia

Spatial and Epistemic Modalities in Constraint-Based Process Calculi

Sophia Knight[2], Catuscia Palamidessi[2], Prakash Panangaden[3], Frank D. Valencia[1]

[1] CNRS and LIX École Polytechnique de Paris
[2] INRIA and LIX École Polytechnique de Paris
[3] School of Computer Science, McGill University

Abstract. We introduce spatial and epistemic process calculi for reasoning about spatial information and knowledge distributed among the agents of a system. We introduce domain-theoretical structures to represent spatial and epistemic information. We provide operational and denotational techniques for reasoning about the potentially infinite behaviour of spatial and epistemic processes. We also give compact representations of infinite objects that can be used by processes to simulate announcements of common knowledge and global information.

Introduction. Distributed systems have changed substantially in the recent past with the advent of phenomena like social networks and cloud computing. In the previous incarnation of distributed computing [16] the emphasis was on consistency, fault tolerance, resource management and related topics; these were all characterized by *interaction between processes*. Research proceeded along two lines: the algorithmic side which dominated the Principles Of Distributed Computing conferences and the more process algebraic approach epitomized by CONCUR where the emphasis was on developing compositional reasoning principles. What marks the new era of distributed systems is an emphasis on managing access to information to a much greater degree than before.

Epistemic concepts were crucial in distributed computing as was realized in the mid 1980s with Halpern and Moses' groundbreaking paper on common knowledge [13]. This led to a flurry of activity in the next few years [11] with many distributed protocols being understood from an epistemic point of view. The impact of epistemic ideas in the concurrency theory community was slower in coming. In an invited talk by one of us [20] at a joint PODC-CONCUR conference in 2008, this point was emphasized and a plea was made for epistemic ideas to be exploited more by concurrency theorists.

The goal of the present paper is simple: to put epistemic concepts in the hands of programmers rather than just appearing in post-hoc theoretical analyses. One could imagine the incorporation of these ideas in a variety of process algebraic settings – and indeed we expect that such formalisms will appear in due course – but what is particularly appealing about the *concurrent constraint programming (ccp)* paradigm [24, 25] is that it was designed to give programmers explicit access to the concept of *partial information* and, as such, had close ties with logic [21, 18]. This makes it ideal for the incorporation of epistemic concepts by expanding the logical connections to include modal logic [15]. In particular, agents posting and querying information in the presence of *spatial hierarchies for sharing information and knowledge*, e.g. friend circles and

shared albums in social networks or shared folders in cloud storage, provide natural examples of managing information access. These domains raise important problems such as the design of models to predict and prevent privacy breaches, which are commonplace nowadays.

Contributions. In ccp [24, 25] processes interact with each other by querying and posting information to a single centralized shared-store. The information and its associated partial order are specified as a *constraint system*, which can be seen as a *Scott information system* without consistency structure [1]. The centralized notion of store, however, makes ccp unsuitable for systems where information and processes can be shared or spatially distributed among certain groups of agents. In this paper we enhance and generalize the theory of ccp for systems with spatial distribution of information.

In Section 1 we generalize the underlying theory of constraint systems by adding *space functions* to their structure. These functions can be seen as topological and closure operators and they provide for the specification of spatial and epistemic information. In Section 2 we extend ccp with a *spatial/epistemic* operator. The spatial operator can specify a process, or a local store of information, that resides within the *space* of a given agent (e.g., an application in some user's account, or some private data shared with a specific group). This operator can also be interpreted as an *epistemic* construction to specify that the information computed by a process will be known to a given agent. It is crucial that one make the distinction between agent and process. The processes are programs, they are mindless and do not "know" anything; the agents are other primitive entities in our model that can be viewed as spatial locations (a passive view) or as active entities that control a locus of information and interact with the global system by launching processes.

It also worth noticing that the ccp concept of local variables cannot faithfully model what we are calling local spaces, since in our spatial framework we can have inconsistent local stores without propagating their inconsistencies towards the global store.

In Section 3 we give a natural notion of observable behaviour for spatial/epistemic processes. Recursive processes are part of our framework, accordingly the notion of observable may involve limits of the spatial information in fair, possibly infinite, computations. These limits may result in infinite or, more precisely, *non-compact* objects involving unbounded nestings of spaces, or epistemic specifications such as *common knowledge*. We then provide a finitary characterization of these observables avoiding complex concepts such as fairness and limits. We also provide a compositional denotational characterization of the observable behaviour. Finally, in Section 4 we address the technical issue of giving *finite approximations* of non-compact information. (An extended version of this work is at http://www.lix.polytechnique.fr/∼fvalenci/papers/eccp-extended.pdf.)

1 Space and Knowledge in Constraint Systems

In this section we introduce two new notions of constraint system for reasoning about distributed information and knowledge in ccp. We presuppose basic knowledge of domain theory and modal logic [1, 22].

Flat Constraint Systems. The ccp model is parametric in a *constraint system (cs)* specifying the structure and interdependencies of the information that processes can

ask of and add to a *central shared store*. This information is represented as assertions traditionally referred to as *constraints*. Following [8, 21] we regard a cs as a complete algebraic lattice in which the ordering $\sqsubseteq$ is the reverse of an entailment relation: $c \sqsubseteq d$ means d *entails* c, i.e., d contains "more information" than c. The top element $false$ represents inconsistency, the bottom element $true$ is the empty constraint, and the *least upper bound* (lub) $\sqcup$ is the join of information. $\bigsqcup S$ is the lub of the elements in S.

Definition 1 (cs). *A* constraint system (cs) $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false)$ *is a complete algebraic lattice where* Con*, the set of constraints, is a partially ordered set wrt* $\sqsubseteq$*,* Con_0 *is the subset of* compact *elements of* Con*,* $\sqcup$ *is the lub operation defined on all subsets, and* $true$*,* $false$ *are the least and greatest elements of* Con*, respectively.*

Remark 1. Recall that $\mathbf{C}$ is a *complete lattice* iff each subset of Con has a least upper bound in Con. Also $c \in Con$ is *compact (finite)* iff for any directed subset D of Con, $c \sqsubseteq \bigsqcup D$ implies $c \sqsubseteq d$ for some $d \in D$. $\mathbf{C}$ is *algebraic* iff for each $c \in Con$, the set of compact elements below it forms a directed set and the lub of this directed set is c.

Example 1. We briefly explain the *Herbrand* cs from [24, 25]. This cs captures *syntactic* equality between terms $t, t', \ldots$ built from a first-order alphabet $\mathcal{L}$ with countably many variables $x, y, \ldots$, function symbols, and equality $=$. The constraints are sets of equalities over the terms of $\mathcal{L}$ (e.g., $\{x = t, y = t\}$ is a constraint). The relation $c \sqsubseteq d$ holds if the equalities in c follow from those in d (e.g., $\{x = y\} \sqsubseteq \{x = t, y = t\}$). The constraint $false$ is the set of all term equalities in $\mathcal{L}$ and $true$ is (the equivalence class of) the empty set. The compact elements are the (equivalence clases of) finite sets of equalities. The lub is (the equivalence class of) set union. (See [24, 25] for full details).

Spatial Constraint Systems. A crucial issue in distributed and multi-agent scenarios is that agents may have their own space for their local information or for performing their computations. We shall address this issue by introducing a notion of space for agents. In our approach each agent i has a *space* $\mathfrak{s}_i$. We can then think of $\mathfrak{s}_i(c)$ as an assertion stating that c holds *within a space attributed to agent* i. Thus, given a store $s = \mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) \sqcup e$ we may think of c and d as holding within the spaces that agents i and j have in s, respectively. Similarly, $\mathfrak{s}_i(\mathfrak{s}_j(c))$ can be viewed as a hierarchical spatial specification stating that c holds within the space the agent i attributes to agent j.

An n-agent *spatial constraint system (n-scs)* is a cs parametric in n structure-preserving constraint mappings $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$ capturing the above intuitions.

Definition 2 (scs). *An* n*-agent* spatial constraint system (n-scs) $\mathbf{C}$ *is a cs equipped with* n *lub and bottom preserving maps* $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$ *over its set of constraints* Con*. More precisely, each* $\mathfrak{s}_i : Con \to Con$ *must satisfy the following properties: (*S.1*)* $\mathfrak{s}_i(true) = true$*, and (*S.2*)* $\mathfrak{s}_i(c \sqcup d) = \mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)$.

Henceforth, given an n*-scs* $\mathbf{C}$*, we refer to each* $\mathfrak{s}_i$ *as the* space *(function) of agent* i *in* $\mathbf{C}$*. We use* $(Con, Con_0, \sqsubseteq, \sqcup, true, false, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$ *to denote the corresponding* n*-scs with space functions* $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$*. We shall simply write "scs" when* n *is unimportant.*

Intuitively, S.1 states that having an empty local store amounts to nothing and S.2 allows us to join pieces of information of agent i. From S.2 one can draw the immediate inference that space functions are monotone: Property S.3 below says that if c can be derived from d then any agent should be able to derive c from d within its own space.

Corollary 1. *Let* $\mathbf{C}$ *be an* n*-scs with space functions* $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$*. Then for each* $\mathfrak{s}_i$ *the following property holds: (*S.3*) If* $c \sqsubseteq d$ *then* $\mathfrak{s}_i(c) \sqsubseteq \mathfrak{s}_i(d)$.

Inconsistency Confinement. In an scs nothing prevents us from having $\mathfrak{s}_i(\mathit{false}) \neq \mathit{false}$. Intuitively, inconsistencies generated by an agent may be confined within its own space. It is also possible to have $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) \neq \mathit{false}$ even when $c \sqcup d = \mathit{false}$; i.e. we may have agents whose information is inconsistent with that of other agents. This reflects the distributive nature of the agents as they may have different information about the same incident. The following notions capture the above-mentioned situations.

Definition 3. *An* n*-scs* $\mathbf{C} = (\mathit{Con}, \mathit{Con}_0, \sqsubseteq, \sqcup, \mathit{true}, \mathit{false}, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$ *is said to be* (i, j) space-consistent *wrt* (c, d) *iff* $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) \neq \mathit{false}$*. Also,* $\mathbf{C}$ *is said to be* (i, j) space-consistent *iff it is* (i, j) *space-consistent wrt to each* $(c, d) \in \mathit{Con} \times \mathit{Con}$*. Furthermore,* $\mathbf{C}$ *is space-consistent iff it is* (i, j) *space-consistent for all* $i, j \in \{1, \ldots, n\}$.

We will see an important class of logical structures characterized as space-consistent scs's in Applications (Section 1). From the next proposition we conclude that to check (i, j) space-consistency it is sufficient to verify whether $\mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(\mathit{false}) \neq \mathit{false}$.

Proposition 1. *Let* $\mathbf{C}$ *be an* n*-scs with space functions* $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$*. Then (1)* $\mathbf{C}$ *is* (i, j) *space-consistent if* $\mathfrak{s}_i(\mathit{false}) \sqcup \mathfrak{s}_j(\mathit{false}) \neq \mathit{false}$ *and (2) if* $\mathbf{C}$ *is* (i, j) *space-consistent then* $\mathfrak{s}_i(\mathit{false}) \neq \mathit{false}$.

Distinctness preservation. Analogous to inconsistency confinement, we could have $\mathfrak{s}_i(c) = \mathfrak{s}_i(d)$ for $c \neq d$. Depending on the intended model this could be interpreted as saying that agent i cannot distinguish c from d. For some applications, however, one may require the space functions to preserve distinctness

Definition 4. *An* n*-scs* $\mathbf{C}$ preserves distinctness *iff all its space functions are injective.*

Shared and Global Information. We conclude by introducing a lub construction that captures the intuition that a given constraint holds in a shared space and globally.

Definition 5. *Let* $\mathbf{C}$ *be an* n*-scs with space functions* $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$. Group-spaces $\mathfrak{s}_G(\cdot)$ *and* global *information* $\mathfrak{g}_G(\cdot)$ *of* $G \subseteq \{1, \ldots, n\}$ *are defined thus:* $\mathfrak{s}_G(c) = \bigsqcup_{i \in G} \mathfrak{s}_i(c)$ *and* $\mathfrak{g}_G(c) = \bigsqcup_{j=0}^{\infty} \mathfrak{s}_G^j(c)$*, where* $\mathfrak{s}_G^0(c) = c$ *and* $\mathfrak{s}_G^{k+1}(c) = \mathfrak{s}_G(\mathfrak{s}_G^k(c))$.

The constraint $\mathfrak{g}_G(c)$ is easily seen to entail c and $\mathfrak{s}_{i_1}(\mathfrak{s}_{i_2}(\ldots(\mathfrak{s}_{i_m}(c))\ldots))$ for any $\{i_1, \ldots, i_m\} \subseteq G$. Thus it realizes the intuition that c holds *globally* wrt G: c holds in each nested space involving only the agents in G.

Epistemic Constraint Systems. We now wish to use $\mathfrak{s}_i(c)$ to represent not only some information c that agent i has but rather a *fact* that he knows. In this case, (i, j)-space consistency wrt any pair of inconsistent information (Definition 3) would not be considered admissible. For in epistemic reasoning if an agent knows facts, those facts must be true, hence asserting that an agent i knows false or inconsistent information would be a fallacy. Thus, $\mathfrak{s}_i(\mathit{false}) = \mathit{false}$ and $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) = \mathit{false}$ if $c \sqcup d = \mathit{false}$.

The domain theoretical nature of constraint systems allows for a rather simple and elegant characterization of knowledge by requiring our space functions to be *Kuratowski closure operators* [17]: i.e., lub and bottom preserving closure operators.

Definition 6 **(n-ecs).** *An n-agent* epistemic constraint system (n-ecs) $\mathbf{C}$ *is an n-scs whose space functions $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$ are also* closure operators*. Thus, in addition to* S.1, S.2 *in Def. 2, each $\mathfrak{s}_i$ also satisfies: (*E.1*) $c \sqsubseteq \mathfrak{s}_i(c)$ and (*E.2*) $\mathfrak{s}_i(\mathfrak{s}_i(c)) = \mathfrak{s}_i(c)$.*

Intuitively, in an n-ecs, $\mathfrak{s}_i(c)$ states that the agent i has knowlege of c in its store $\mathfrak{s}_i$. The axiom E.1 says that if agent i knows c then c must hold, hence $\mathfrak{s}_i(c)$ has at least as much information as c. The epistemic principle that an agent i is aware of its own knowledge (the agent knows what he knows) is realized by E.2. Also the epistemic assumption that agents are idealized reasoners follows from S.3 in Corollary 1; for if c is a consequence of d ($c \sqsubseteq d$) then if d is known to agent i, so is c, $\mathfrak{s}_i(c) \sqsubseteq \mathfrak{s}_i(d)$.

Common Knowledge. Epistemic constructions such as "the agent i knows that agent j knows c" can be expressed as $\mathfrak{s}_i(\mathfrak{s}_j(c))$. The *group knowledge* of a fact c in a group of agents G happens when all the agents in G know c. This can be represented as $\mathfrak{s}_G(c)$ in Definition 5. Similarly, *common knowlege* of a fact c in a group G happens when all the agents in G know c, they all know that they know c, and so on ad infinitum. This can be captured by the lub construction $\mathfrak{g}_G(c)$ in Definition 5.

Remark 2. Consider an n-ecs $\mathbf{C}$ whose compact elements Con_0 are closed under the space functions: i.e., if $c \in Con_0$ the $\mathfrak{s}_i(c) \in Con_0$. Clearly Con_0 is closed under group knowledge $\mathfrak{s}_G(c)$ since G is finite. It is not necessarily closed under common knowledge $\mathfrak{g}_G(c)$ because, in general, $\bigsqcup_{j=1}^{\infty} \mathfrak{s}_G^j(c)$ cannot be finitely approximated. Nevertheless, in Applications (Section 1) we shall identify families of scs's where Con_0 is closed under common knowledge, and in Section 4 we address the issue of using suitable over-approximations of common knowledge.

The following proposition states two distinctive properties of ecs's: They are not space-consistent, as argued above, and those whose space function is not the identity do not preserve distinctness. We use *id* for the identity space function.

Proposition 2. *Let $\mathbf{C}$ be an n-ecs with space functions $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$. For each $i, j \in \{1, \ldots, n\}$: (1) $\mathbf{C}$ is not (i, j)-space consistent, and (2) if $\mathfrak{s}_i \neq id$ then $\mathfrak{s}_i$ is not injective.*

Applications. We shall now illustrate important families of scs's. The families reveal meaningful connections between our scs's and models of knowledge and belief [11].

Aumann Constraint Systems. Aumann structures [11] are an alternative *event-based* approach to modelling knowledge. An Aumann structure is a tuple $\mathcal{A} = (S, \mathcal{P}_1, ..., \mathcal{P}_n)$ where S is a set of states and each $\mathcal{P}_i$ is a partition on S for agent i. We call these partitions *information sets*. If two states t and u are in the same information set for agent i, it means that in state t agent i considers state u possible, and vice versa. An *event* in an Aumann structure is any subset of S. Event e holds at state t if $t \in e$. The conjunction of two events is their intersection and knowledge operators are defined as $\mathsf{K}_i(e) = \{t \in S \mid \mathcal{P}_i(t) \subseteq e\}$ where $\mathcal{P}_i(t)$ denotes the set where t appears in $\mathcal{P}_i$.

We define the Aumann n-ecs $\mathbf{C}(\mathcal{A})$ as follows: The constraints are the events, i.e., $Con = \{e \mid e \text{ is an event in } \mathcal{A}\}$, $e_1 \sqsubseteq e_2$ iff $e_2 \subseteq e_1$, $\sqcup$ is the set intersection of two events, $true$ is the event containing every state in S, and $false$ is the event containing no states. The space function for each agent i is given by $\mathfrak{s}_i(e) = \mathsf{K}_i(e)$. $\square$

Theorem 1. *For any Aumann structure $\mathcal{A} = (S, \mathcal{P}_1, ..., \mathcal{P}_n)$, $\mathbf{C}(\mathcal{A})$ is an n-ecs.*

Aumann constraint systems are ecs's, thus they are not space-consistent (Proposition 2). We shall now identify a meaningful scs that is space-consistent.

Kripke Constraint Systems. A Kripke structure can be seen as a labeled transition system (LTS) where the labels represent agents and the transitions represent accessibility relations for the agents: if $s \xrightarrow{i} t$ then in state s, agent i considers t possible. An epistemic Kripke structure is an LTS where the transition relations are equivalences. In the following scs, the constraints are sets of *pointed Kripke structures*, i.e., sets of pairs (M, s) where M is a Kripke structure and s is a state of M.

Consider a set of Kripke structures $\mathfrak{M}$ over agents $\{1, ..., n\}$. Let $\Delta_{\mathfrak{M}}$ be the set $\{(M, t) \mid M \in \mathfrak{M} \text{ and } t \in St(M)\}$ where $St(M)$ denotes the set of states of M. Define an n-scs $\mathbf{C}(\Delta_{\mathfrak{M}})$ as follows: Let $Con = \mathcal{P}(\Delta_{\mathfrak{M}})$ and $c_1 \sqsubseteq c_2$ iff $c_2 \subseteq c_1$. This generates a complete algebraic lattice, where $c_1 \sqcup c_2$ is the set intersection of c_1 and c_2. The compact elements of the lattice are the cofinite sets, that is, if $\Delta_{\mathfrak{M}} \backslash c$ is a finite set, then c is a compact element in the lattice. Finally, define $\mathfrak{s}_i(c) = \{(M, t) \mid \forall t' \in St(M) \left[t \xrightarrow{i}_M t' \implies (M, t') \in c\right]\}$– this definition is reminiscent of the semantics of the box modality in modal logic [22]. □

The following theorem gives us a taxonomy of scs's for the above construction.

Theorem 2. *For any non-empty set of Kripke structures $\mathfrak{M}$ over agents $\{1, ..., n\}$, (1) $\mathbf{C}(\Delta_{\mathfrak{M}})$ is an n-scs, (2) if $\mathfrak{M}$ is the set of all pointed Kripke structures, $\mathbf{C}(\Delta_{\mathfrak{M}})$ is a space-consistent n-scs, and (3) if $\mathfrak{M}$ is the set of all pointed Kripke structures whose accessibility relations are equivalences then $\mathbf{C}(\Delta_{\mathfrak{M}})$ is an n-ecs.*

Remark 3. Consider the modal formulae given by $\phi := p \mid \phi \wedge \phi \mid \Box_i \phi$, where p is a basic proposition, and the corresponding usual notion of satisfaction over Kripke models for propositions, conjunction and the box modality (see [22]). We abuse the notation and use a formula ϕ to denote the set of all pointed Kripke structures that satisfy ϕ. With the help of the above theorem, one can establish a correspondence between the n-scs satisfying the premise in (2) and the modal system K_n [11] in the sense that ϕ is above ϕ' in the lattice iff we can derive in K_n that ϕ implies ϕ' (written $\vdash_{K_N} \phi \Rightarrow \phi'$). Similarly, for the n-scs satisfying (3) and the epistemic system $S4_n$ [11].

We conclude by giving sufficient conditions for compactness of the constraints in $\mathbf{C}(\Delta_{\mathfrak{M}})$. The compact elements of $\mathbf{C}(\Delta_{\mathfrak{M}})$ are the cofinite subsets of $\Delta_{\mathfrak{M}}$. If $\Delta_{\mathfrak{M}}$ is a finite set (this occurs if $\mathfrak{M}$ is a finite set of finite state Kripke structures), then every subset of $\Delta_{\mathfrak{M}}$ is cofinite, and therefore each element of the lattice is compact, even $\mathfrak{g}_G(c)$ (Remark 2). Thus, if $\Delta_{\mathfrak{M}}$ is finite then each constraint in $\mathbf{C}(\Delta_{\mathfrak{M}})$ is compact.

2 Space and Knowledge in Processes

We now introduce two ccp variants: *spatial ccp (sccp)* and *epistemic ccp (eccp)*. The former is a conservative extension of ccp to model agents with spaces, possibly nested, in which they can store information and run processes. Its underlying cs is an scs. The

latter extends the former with additional rules to model agents that interact by asking and computing knowledge within the spatial information distribution. Its underlying scs is an ecs. For semantic reasons, we require our scs be continuous and space-compact.

Definition 7. *An n-scs $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$ is said to be* continuous *iff for every directed set $S \subseteq Con$ and every $\mathfrak{s}_i$, $\mathfrak{s}_i(\bigsqcup S) = \bigsqcup_{e \in S} \mathfrak{s}_i(e)$. Furthermore $\mathbf{C}$ is said to be* space-compact *iff for every $\mathfrak{s}_i$, $\mathfrak{s}_i(c) \in Con_0$ if $c \in Con_0$.*

Our examples (Applications, Section 1) can be shown to be continuous. Aumann ecs's are space-compact under the additional condition that every set in each partition is finite. A Kripke scs is space-compact if the inverse of the accessibility relation is finitely-branching. In the special case of Kripke ecs's this is the same as requiring each agent's accessibility relation to be finitely-branching since these relations are reflexive.

Syntax. The following syntax of processes will be common to both calculi. [4]

Definition 8. *Let $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$ be a continuous and space compact n-scs. Let $A = \{1, ..., n\}$ be the set of agents. Assume a countable set of variables $Vars = \{X, Y, \ldots\}$. The terms are given by the following syntax:*

$$P, Q \ldots ::= \mathbf{0} \mid \mathbf{tell}(c) \mid \mathbf{ask}(c) \rightarrow P \mid P \parallel Q \mid [P]_i \mid X \mid \mu X.P$$

where $c \in Con_0, i \in A$, and $X \in Vars$. A term T is said to be closed iff every variable X in T occurs in the scope of an expression $\mu X.P$. We shall refer to closed terms as processes *and use Proc to denote the corresponding set.*

Before giving semantics to our processes, we give some intuitions about their behaviour. The *basic processes* are tell, ask, and parallel composition and they are defined as in standard ccp [25]. Intuitively, $\mathbf{tell}(c)$ in a store d adds c to d to make c available to other processes with access to this store. This addition, represented as $d \sqcup c$, is performed whether or not $d \sqcup c = \mathit{false}$. The process $\mathbf{ask}(c) \rightarrow P$ in a store e may execute P if c is entailed by e, i.e., $c \sqsubseteq e$. The process $P \parallel Q$ stands for the *parallel execution* of P and Q. The following example will be referred to throughout the paper.

Example 2. Let us take $P = \mathbf{tell}(c)$ and $Q = \mathbf{ask}(c) \rightarrow \mathbf{tell}(d)$. From the above intuitions, it follows that in $P \parallel Q$ both c and d will be added to the store.

Spatial Processes. Our spatial ccp variant can be thought of as a *shared-spaces* model of computation. Each agent $i \in A$ may have computational spaces of the form $[\cdot]_i$ where processes as well as other agents' spaces may reside. It also has a space function $\mathfrak{s}_i$ representing the information stored in its spaces. Recall that $\mathfrak{s}_i(c)$ states that c holds in the space of agent i. Similarly, $\mathfrak{s}_i(\mathfrak{s}_j(c))$ means that c holds in the store that agent j has within the space of agent i. Unlike any other ccp calculus, it is possible to have agents with inconsistent information since $c \sqcup d = \mathit{false}$ does not necessarily imply $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) = \mathit{false}$ (see space-consistent ecs in Definition 3).

The spatial construction $[P]_i$ represents a process P running within the space of agent i. Any information c that P produces is available to processes that lie within the same space. We shall use $[P]_G$, where $G \subseteq A$, as an abbreviation of $\|_{i \in G}\ [P]_i$.

[4] For the sake of space and clarity, we dispense with the local/hiding operator.

Example 3. Consider $[P]_i \parallel [Q]_i$ with P and Q as in Ex. 2. From the above intuitions it follows that both c and d will be added to store of agent i, i.e., we will have $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)$. Similarly, $[P \parallel Q]_i$ will produce $c \sqcup d$ in the store of agent i, i.e., $\mathfrak{s}_i(c \sqcup d)$ which from the scs axioms is equivalent to $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)$. In fact, we will equate the behaviour of $[P]_i \parallel [Q]_i$ with that of $[P \parallel Q]_i$. In $[P]_j \parallel [Q]_i$ for $i \neq j$, d will not necessarily be added to the space of i because c is not made available for agent i. Also in $P \parallel [Q]_i$, d is not added to the the space of i. In this case, however, we may view the c told by P as being available at an outermost space that *does not belong to any agent*. This does not mean that c holds everywhere, i.e., *globally* (Def. 5). Finally, consider $[P]_{\{i,j\}} \parallel [[Q]_i]_j$. Here d will not necessarily be added to the space agent i has within the space of agent j because in an scs although $\mathfrak{s}_i(c)$ and $\mathfrak{s}_j(c)$ hold, $\mathfrak{s}_j(\mathfrak{s}_i(c))$ may not hold.

Epistemic Processes. For our epistemic ccp variant, we shall further require that the underlying scs be epistemic, i.e., an ecs. This gives the operator $[P]_i$ additional behaviour. From an epistemic point of view, the information c produced by P not only becomes available to agent i, as in the spatial case, but also it becomes a *fact*. This does not necessarily mean, of course, that c will be available everywhere, as there are facts that some agents may not know. It does mean, however, that unlike the spatial case, we cannot allow agents' spaces to include inconsistent information, as facts *cannot be contradictory*–in an ecs, $c \sqcup d = \mathit{false}$ implies $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) = \mathit{false}$.

Operationally, $[P]_i$ causes any information c produced by P to become available not only in the space of agent i but also in any space in which $[P]_i$ is included. This is because epistemically $\mathfrak{s}_i(c) = c \sqcup \mathfrak{s}_i(c)$ so if $\mathfrak{s}_j(\mathfrak{s}_i(c))$ holds, then $\mathfrak{s}_j(c \sqcup \mathfrak{s}_i(c))$ also holds, and similarly $c \sqcup \mathfrak{s}_j(c \sqcup \mathfrak{s}_i(c))$. This can be viewed as saying that c propagates outward in space.

Example 4. Consider $[Q \parallel [P]_i]_j$ with P and Q as in Example 2. Notice that from executing P we obtain $\mathfrak{s}_j(\mathfrak{s}_i(c))$. In the spatial case, Q will not necessarily tell d because in an scs, $\mathfrak{s}_j(\mathfrak{s}_i(c))$ may not entail $\mathfrak{s}_j(c)$. On the other hand, in the epistemic case, Q will tell d since in any ecs, $\mathfrak{s}_j(\mathfrak{s}_i(c)) = \mathfrak{s}_j(c \sqcup \mathfrak{s}_i(c))$ which entails $\mathfrak{s}_j(c)$.

Infinite Processes. Unbounded behaviour is specified using recursive definitions of the form $\mu X.P$ whose behaviour is that of $P[\mu X.P/X]$, i.e., P with every free occurrence of X replaced with $\mu X.P$. We assume that recursion is *ask guarded*: i.e., for every $\mu X.P$, each occurrence of X in P occurs under the scope of an ask process. For simplicity we assume an implicit "$\mathbf{ask}(\mathit{true}) \rightarrow$ " in unguarded occurrences of X.

Recursive definitions allow us to define complex spatial and epistemic situations. Given $G \subseteq A$ and a basic process P we define $\mathit{global}(G, P) \stackrel{\text{def}}{=} P \parallel \mu X.\, [P \parallel X]_G$. Intuitively, in $\mathit{global}(G, P)$ any information c produced by P will be available at any space or any nesting of spaces involving only the agents in G. Consider the process $\mathit{global}(G, P) \parallel \left[[\ldots [Q]_{k_m} \ldots]_{k_2}\right]_{k_1}$ where $G = \{k_1, ..., k_m\} \subseteq A$, with P and Q as in Example 2. The process $\mathit{global}(G, P)$ eventually makes c available in the (nested) space $\left[[\ldots [\cdot]_{k_m} \ldots]_{k_2}\right]_{k_1}$ and thus Q will tell d in that space.

Spatial and Epistemic Reduction Semantics. We now define a structural operational semantics (sos) for sccp and eccp. We begin with the sos for the spatial case. The sos

$$\mathbf{T}\langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{0}, d \sqcup c\rangle \qquad \mathbf{A}\frac{c \sqsubseteq d}{\langle \mathbf{ask}\ (c)\ \rightarrow\ P, d\rangle \longrightarrow \langle P, d\rangle} \qquad \mathbf{PL}\frac{\langle P, d\rangle \longrightarrow \langle P', d'\rangle}{\langle P \parallel Q, d\rangle \longrightarrow \langle P' \parallel Q, d'\rangle}$$

$$\mathbf{R}\frac{\langle P[\mu X.P/X], d\rangle \longrightarrow \gamma}{\langle \mu X.P, d\rangle \longrightarrow \gamma} \qquad \mathbf{S}\frac{\langle P, c^i\rangle \longrightarrow \langle P', c'\rangle}{\langle [P]_i, c\rangle \longrightarrow \langle [P']_i, c \sqcup \mathfrak{s}_i(c')\rangle} \qquad \mathbf{E}\frac{\langle P, c\rangle \longrightarrow \langle P', c'\rangle}{\langle [P]_i, c\rangle \longrightarrow \langle [P]_i \parallel P', c'\rangle}$$

Table 1. Rules for sccp and eccp (see Convention 1). The projection c^i is given in Definition 9. The symmetric right rule for PL, PR, is omitted. Rule E only applies to eccp.

for the epistemic case extends the spatial one with an additional rule and the assumption that the underlying scs is an ecs. Henceforth we shall use the following convention:

Convention 1 *The relations in following sections assume an underlying continuous and space-compact n-scs $\mathbf{C} = (Con, Con_0, \sqsubseteq, \sqcup, true, false, \mathfrak{s}_1, \ldots, \mathfrak{s}_n)$. We sometimes index them with "$\mathtt{s}$" if they are interpreted for sccp, and with "$\mathtt{e}$" if they are interpreted for eccp. We often omit the indexes when they are irrelevant or obvious.*

A configuration is a pair $\langle P, c\rangle \in Proc \times Con$ where c represents the current *spatial distribution* of information in P. We use $Conf$ with typical elements $\gamma, \gamma', \ldots$ to denote the set of configurations. The sos for sccp is given by means of the transition relation between configurations $\longrightarrow_{\mathtt{s}} \subseteq Conf \times Conf$ obtained by replacing $\longrightarrow$ with $\longrightarrow_{\mathtt{s}}$ in the rules A, T, PL (and its symmetric version), R, and S in Table 1.

The rules A, T, PL, and R for the basic processes and recursion are standard in ccp and they are easily seen to realize the above intuitions (see [25]). The rule S for the new spatial operator is more involved and we explain it next. First we introduce the following central notion defining the projection of a spatial constraint c for agent i.

Definition 9 (Views). *The agent i's* view *of c, c^i, is given by $c^i = \bigsqcup\{d \mid \mathfrak{s}_i(d) \sqsubseteq c\}$.*

Intuitively, c^i represents all the information the agent i may see or have in c. For example if $c = \mathfrak{s}_i(d) \sqcup \mathfrak{s}_j(e)$ then agent i sees d, so $d \sqsubseteq c^i$. Observe that if $\mathfrak{s}_i(d) = \mathfrak{s}_i(d')$ then $(\mathfrak{s}_i(d))^i$ entails both d and d'. This is intended because $\mathfrak{s}_i(d) = \mathfrak{s}_i(d')$ means that agent i cannot distinguish d from d'. The constraint c^i enjoys the following property which will be useful later on.

Lemma 1. *For any constraint c, $c \sqcup \mathfrak{s}_i(c^i) = c$.*

Let us now describe the rule S for the spatial operator. First, in order for $[P]_i$ with store c to make a reduction, the information agent i sees or has in c must allow P to make the reduction. Hence we run P with store c^i. Second, the information d that P's reduction would add to c^i is what $[P]_i$ adds to the space of agent i as stated in Proposition 3 below.

Proposition 3. *If $\langle P, c^i\rangle \longrightarrow \langle P', c^i \sqcup d\rangle$ then $\langle [P]_i, c\rangle \longrightarrow \langle [P']_i, c \sqcup \mathfrak{s}_i(d)\rangle$.*

Next we show an instructive reduction involving the use of the S rule.

Example 5. Take $R = [P]_i \parallel [Q]_i$ with P and Q as in Example 2. One can verify that $\langle R, true\rangle \longrightarrow \langle [\mathbf{0}]_i \parallel [Q]_i, \mathfrak{s}_i(c)\rangle \longrightarrow \langle [\mathbf{0}]_i \parallel [\mathbf{0}]_i, \mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d)\rangle$. Recall that $\mathfrak{s}_i(c) \sqcup \mathfrak{s}_i(d) = \mathfrak{s}_i(c \sqcup d)$. A more interesting example is $T = [\mathbf{tell}(c')]_i \parallel [Q]_i$ under the assumption that $\mathfrak{s}_i(c) = \mathfrak{s}_i(c')$. We have $\langle T, true\rangle \longrightarrow \langle [\mathbf{0}]_i \parallel [Q]_i, \mathfrak{s}_i(c')\rangle \longrightarrow \langle [\mathbf{0}]_i \parallel [\mathbf{0}]_i, \mathfrak{s}_i(c') \sqcup \mathfrak{s}_i(d)\rangle$. d is told by Q within the space of i because $\mathfrak{s}_i(c) = \mathfrak{s}_i(c')$, so c and c' are regarded as equivalent by i.

Epistemic Semantics. The eccp sos assumes that the underlying scs is an ecs. As explained earlier given $[P]_i$, the information c produced by P not only becomes available to agent i but also becomes a *fact* within the hierarchy of spaces in which $[P]_i$ is included. This means that c is available not only in the space of agent i but also in any space in which $[P]_i$ is included. We can view this as saying that c propagates *outwards* through the spaces $[P]_i$ is in and this is partly realized by the equation $\mathfrak{s}_i(c) = c \sqcup \mathfrak{s}_i(c)$ which follows from E.1 in ecs (Definition 6). Mirroring this constraint equation and epistemic reasoning, the behaviour of $[P]_i$ and $P \parallel [P]_i$ must also be equated (since P can only produce factual information). This makes $[P]_i$ reminiscent of the replication/bang operator in the π-calculus [19]. For eccp we include Rule E in Table 1. As illustrated in Example 6, Rule E is necessary for the behaviour of $[P]_i$ and $P \parallel [P]_i$ to be the same, corresponding to the epistemic principles we wish to mimic.

The sos of eccp is given by the transition relation between configurations $\longrightarrow_{\mathtt{e}} \subseteq \mathit{Conf} \times \mathit{Conf}$ obtained by replacing $\longrightarrow$ with $\longrightarrow_{\mathtt{e}}$ in the rules in Table 1 and assuming the underlying scs to be an ecs.

Example 6. Let $R = [P \parallel [Q]_i]_j$ and $T = [P \parallel [Q]_i \parallel Q]_j$ with P and Q as in Example 2. We wish to equate R and T to mimic epistemic principles. Even assuming an ecs, with only the rules of sccp (i.e., without Rule E), T can produce $\mathfrak{s}_j(d)$, d in the store of agent j, but R is not necessarily able to do this: One can verify that there are T', e' s.t. $\langle T, true\rangle \longrightarrow_{\mathtt{s}}^* \langle T', e'\rangle$ and $\mathfrak{s}_j(d) \sqsubseteq e'$, while, in general, for all R', e'' s.t., $\langle R, true\rangle \longrightarrow_{\mathtt{s}}^* \langle R', e''\rangle$ we have $\mathfrak{s}_j(d) \not\sqsubseteq e''$. With the rules of eccp, however, one can verify for each e' s.t. $\langle T, true\rangle \longrightarrow_{\mathtt{e}}^* \langle T', e'\rangle$ there exists e'', $\langle R, true\rangle \longrightarrow_{\mathtt{e}}^* \langle R', e''\rangle$ such that $e' \sqsubseteq e''$ (and vice-versa with the roles of R and T interchanged).

3 Observable Behaviour of Space and Knowledge

A standard notion of observable behaviour in ccp involves infinite fair computations and information constructed as the limit of finite approximations. For our calculi, however, these limits may result in infinite (or non-compact) objects involving arbitrary nesting of spaces, or epistemic specifications such as common knowledge. In this section we provide techniques useful for analyzing the observable behaviour of such processes using simpler finitary concepts and compositional reasoning.[5]

The notion of *fairness* is central to the definition of observational equivalence for ccp. We introduce this notion following [12]. Any derivation of a transition involves an application of Rule A or Rule T. We say that P is *active* in a transition $t = \gamma \longrightarrow \gamma'$ if there exists a derivation of t where rule A or T is used to produce a transition of the

[5] See Convention 1.

form $\langle P, d\rangle \longrightarrow \gamma''$. Moreover, we say that P is *enabled* in γ if there exists γ' such that P is active in $\gamma \longrightarrow \gamma'$. A computation $\gamma_0 \longrightarrow \gamma_1 \longrightarrow \gamma_2 \longrightarrow \ldots$ is said to be *fair* if for each process enabled in some γ_i there exists $j \geq i$ such that the process is active in γ_j.

Observing Limits. A standard notion of observables for ccp is the *results* computed by a process for a given initial store. The result of a computation is defined as the least upper bound of all the stores occurring in the computation, which, thanks to the monotonic properties of our calculi, form an increasing chain. More formally, given a finite or infinite computation ξ of the form $\langle Q_0, d_0\rangle \longrightarrow \langle Q_1, d_1\rangle \longrightarrow \langle Q_2, d_2\rangle \longrightarrow \ldots$ the result of ξ, denoted by $Result(\xi)$, is the constraint $\bigsqcup_i d_i$. In our calculi all fair computations from a configuration have the same result: Let γ be a configuration and let ξ_1 and ξ_2 be two computations of γ. We can show that if ξ_1 and ξ_2 are fair, then $Result(\xi_1) = Result(\xi_2)$. We can then set $Result(\gamma) \stackrel{\text{def}}{=} Result(\xi)$ for any fair computation ξ of γ.

Definition 10. (Observational equivalence) *Let $\mathcal{O} : Proc \to Con_0 \to Con$ be given by $\mathcal{O}(P)(d) = Result(\langle P, d\rangle)$. We say that P and Q are observationally equivalent, written $P \sim_o Q$, iff $\mathcal{O}(P) = \mathcal{O}(Q)$.*

Example 7. The observation we make of the recursive process $global(G, \mathbf{tell}(c))$ on input $true$ is the limit $\mathfrak{g}_G(c)$ (Definition 5). I.e., $\mathcal{O}(global(G, \mathbf{tell}(c)))(true) = \mathfrak{g}_G(c)$.

The relation $\sim_o$ can be shown to be a congruence, i.e., it is preserved under arbitrary contexts. Recall that a context C is a term with a hole $\bullet$, so that replacing it with a process P yields a process term $C(P)$. E.g., if $C = [\bullet]_i$ then $C(\mathbf{tell}(d)) = [\mathbf{tell}(d)]_i$.

Theorem 3. *$P \sim_o Q$ iff for every context C, $C(P) \sim_o C(Q)$.*

Observing Barbs. In the next section we shall show that the above notion of observation has pleasant and useful closure properties like those of basic ccp. Some readers, however, may feel uneasy with observable behaviour involving notions such as *infinite fair* computations and limits, i.e., possibly *infinite* (or non-compact) elements. Nevertheless, we can give a finitary characterization of behavioral equivalence for our calculi, involving only finite computations and compact elements.

A barb is an element of Con_0, i.e., a compact element. We say that $\gamma = \langle P, d\rangle$ *satisfies* the barb c, written $\gamma \downarrow_c$, iff $c \sqsubseteq d$; γ *weakly satisfies* the barb c, written $\gamma \Downarrow_c$, iff there is γ' s.t. $\gamma \longrightarrow^* \gamma'$ and $\gamma' \downarrow_c$. E.g., $\langle \mathbf{tell}(c) \parallel \mathbf{ask}\ c \to [\mathbf{tell}(d)]_i, true\rangle \Downarrow_{\mathfrak{s}_i(d)}$.

Definition 11. *P and Q are* barb equivalent*, written $P \sim_b Q$, iff $\forall d \in Con_0$, $\langle P, d\rangle$ and $\langle Q, d\rangle$ weakly satisfy the same barbs.*

We now establish the correspondence between our process equivalences. First we recall some facts from domain theory central to our proof of the correspondence. Two (possibly infinite) chains $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ and $e_0 \sqsubseteq e_1 \sqsubseteq \cdots \sqsubseteq e_n \sqsubseteq \ldots$ are said to be *cofinal* if for all d_i there exists an e_j such that $d_i \sqsubseteq e_j$ and vice versa.

Lemma 2. *Let $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ and $e_0 \sqsubseteq e_1 \sqsubseteq \cdots \sqsubseteq e_n \sqsubseteq \ldots$ be two chains. (1) If they are cofinal, then they have the same limit, i.e., $\bigsqcup d_i = \bigsqcup e_i$. (2) If all elements of both chains are* compact *and $\bigsqcup d_i = \bigsqcup e_i$, then the two chains are cofinal.*

DX $[\![X]\!]_I = I(X)$ **DP** $[\![P \parallel Q]\!]_I = [\![P]\!]_I \cap [\![Q]\!]_I$ **D0** $[\![0]\!]_I = \{d \mid d \in Con\}$
DT $[\![\mathbf{tell}(c)]\!]_I = \{d \mid c \sqsubseteq d\}$ **DA** $[\![\mathbf{ask}(c) \rightarrow P]\!]_I = \{d \mid c \sqsubseteq d \text{ and } d \in [\![P]\!]_I\} \cup \{d \mid c \not\sqsubseteq d\}$
DR $[\![\mu X.P]\!]_I = \bigcap\{S \subseteq \mathcal{P}(Con) \mid [\![P]\!]_{I[X:=S]} \subseteq S\}$
DS $[\![[P]_i]\!]_I = \{d \mid d^i \in [\![P]\!]_I\}$ (for sccp) **DE** $[\![[P]_i]\!]_I = \{d \mid d^i \in [\![P]\!]_I\} \cap [\![P]\!]_I$(for eccp)

Table 2. Denotational Equations for sccp and eccp. $I : Var \rightarrow \mathcal{P}(Con)$.

The proof of the correspondence shows that the stores of any pair of *fair* computations of equivalent processes form pairs of cofinal chains. It also uses a relation between weak barbs and fair computations: Let $\langle P_0, d_0\rangle \longrightarrow \langle P_1, d_1\rangle \longrightarrow \ldots \longrightarrow \langle P_n, d_n\rangle \longrightarrow \ldots$ be a fair computation. We can show that if $\langle P_0, d_0\rangle \Downarrow_c$ then there exists a store d_i s.t., $c \sqsubseteq d_i$. With these observations we can show that two processes are not observationally equivalent on a given input iff there is a compact element that tells them apart.

Theorem 4. $\sim_o \; = \sim_b$.

Denotational Semantics. Here we define a denotational characterization of observable behaviour that allows us to reason compositionally about our spatial/epistemic processes. First we can show that the behaviour of a process P, $\mathcal{O}(P)$, is a closure operator on $\sqsubseteq$. The importance of $\mathcal{O}(P)$ being a closure operator on $\sqsubseteq$ is that it is fully determined by its fixed points $\mathit{fix}(\mathcal{O}(P)) = \{d \mid \mathcal{O}(P)(d) = d\}$. More precisely, $\mathcal{O}(P)(c) = \bigsqcup\{d \in Con \mid c \sqsubseteq d \text{ and } d \in \mathit{fix}(\mathcal{O}(P))\}$. Therefore,

Corollary 2. $\mathcal{O}(P) = \mathcal{O}(Q)$ *iff* $\mathit{fix}(\mathcal{O}(P)) = \mathit{fix}(\mathcal{O}(Q))$.

We now give a compositional denotational semantics $[\![\, P \,]\!]$ that captures exactly the set of fixed points of $\mathcal{O}(P)$. More precisely, let I be an assignment function from Var, the set of process variables, to $\mathcal{P}(Con)$. Given a term T, $[\![\, T \,]\!]_I$ is meant to capture the fixed points of T under the assignment I. Notice that if T is a process P, i.e., a closed term, the assignment is irrelevant so we simply write $[\![\, P \,]\!]$. The denotation for processes in sccp is given by the equations DX, D0, DT, DA, DP and DS in Table 2. The denotation for the processes in eccp is given by the same rules except that the rule DS is replaced with the rule DE in Table 2.

The denotations of the basic operators are the same as in standard ccp [25] and are given by equations D0, DT, DA and DP. E.g., DA says that the set of fixed points of $\mathbf{ask}\; c \;\rightarrow\; P$ are those d that do not entail c or that if they do entail c then they are fixed points of P. The denotation of a term X under I is $I(X)$ (see DX). The equation DR for $\mu X.P$ follows from the Knaster-Tarski theorem in the complete lattice $(\mathcal{P}(Con), \subseteq)$.

The denotation of $[P]_i$ in the spatial case is given by equation DS. It says that d is a fixed point for $[P]_i$ if $d^i \in [\![P]\!]$. Recall that d^i is i's view of d, so if $d^i \in [\![P]\!]$, then i's view of d is a fixed point for P. In the operational semantics, the S rule is the only applicable rule for this case. We can use Lemma 1, which says that $d = d \sqcup \mathfrak{s}_i(d^i)$, to prove that if d^i is a fixed point for P then d is a fixed point for $[P]_i$.

The denotation of $[P]_i$ in the epistemic case is given by DE instead of DS. It says that d is a fixed point for $[P]_i$ if $d^i \in [\![P]\!]$, as in the spatial case, and d is fixed point of P. The additional requirement follows from the operational semantics rule E which it amounts to run $[P]_i$ in parallel with (an evolution of) P.

From the above observations we can show that in fact $[\![P]\!] = \mathit{fix}(\mathcal{O}(P))$. Hence, from Corollary 2 we obtain a compositional characterization of observational equivalence, and thus from Theorem 4 also for barb equivalence.

Theorem 5. *$P \sim_o Q$ iff $[\![P]\!] = [\![Q]\!]$.*

4 Compact Approximation of Space and Knowledge

An important semantic property of global information/common knowledge $\mathfrak{g}_G(c)$ (Definition 5) in the underlying scs is that it preserves the *continuity* of the space functions. I.e., one can verify that $\mathfrak{g}_G(\bigsqcup D) = \bigsqcup_{d \in D} \mathfrak{g}_G(d)$ for any directed set $D \subseteq Con$.

In contrast $\mathfrak{g}_G(c)$ does not preserve the *compactness* of the space functions (Remark 2). This means that, although, the limit of infinite computation may produce $\mathfrak{g}_G(c)$, we cannot have a process that refers directly to $\mathfrak{g}_G(c)$ since processes can only ask and tell compact elements. The reason for this syntactic restriction is illustrated below:

Example 8. Suppose we had a process $P = \mathbf{ask}\ \mathfrak{g}_G(c) \rightarrow \mathbf{tell}(d)$ asking whether group G has common knowledge of c and if so posting d. Note that $\mathcal{O}(P)(\mathit{true}) = \mathit{true}$ and $\mathcal{O}(P)(\mathfrak{g}_G(c)) = \mathfrak{g}_G(c) \sqcup d$. Now for $Q = \mathit{global}(G, \mathbf{tell}(c))$ we have $\mathcal{O}(Q)(\mathit{true}) = \mathfrak{g}_G(c)$. But one can verify that $\mathcal{O}(P \parallel Q)(\mathit{true}) = \mathfrak{g}_G(c)$, and thus $\mathcal{O}(P \parallel Q)(\mathcal{O}(P \parallel Q)(\mathit{true})) = \mathcal{O}(P \parallel Q)(\mathfrak{g}_G(c)) = \mathfrak{g}_G(c) \sqcup d$. This would mean that the observation function is not idempotent, contradicting the fact that $\mathcal{O}(P)$ is a closure operator, a crucial property for full abstraction of our denotational semantics.

Nevertheless, asking and telling information of the form $\mathfrak{g}_G(c)$ could be useful in certain protocols to state in one computational step, rather than computing as a limit, common knowledge or global information about certain states of affairs c (e.g., mutual agreement). To address this issue we extend the underlying scs with *compact elements* of the form $\mathfrak{a}_G(c)$ which can be thought of as (over-)approximations of $\mathfrak{g}_G(c)$. The approximation $\mathfrak{a}_G(c)$ can then be used in our processes to simulate the use of $\mathfrak{g}_G(c)$. We refer to $\mathfrak{a}_G(c)$ as a *announcement* of c for the group G to convey the meaning that $\mathfrak{g}_G(c)$ is attained in one step as in a public announcement. We can only define the announcements over a finite subset of compact elements S, since an infinite set would conflict with the continuity $\mathfrak{a}_G(\cdot)$. We only consider announcements for the entire set of agents A (for arbitrary groups the construction follows easily). The above-mentioned extension of an scs $\mathbf{C^1}$ into an scs $\mathbf{C^2}(S)$ with announcement over S is given below:

Definition 12. *Let $\mathbf{C^1} = (Con^1, Con^1_0, \sqsubseteq_1, \mathfrak{s}^1_1, \ldots, \mathfrak{s}^1_n)$ be an scs over agents $A = \{1, \ldots, n\}$. For $S \subseteq_{\mathit{fin}} Con^1_0$, define lattice $\mathbf{C^2}(S) = (Con^2, Con^2_0, \sqsubseteq_2, \mathfrak{s}^2_1, \ldots, \mathfrak{s}^2_n)$ as follows. The set Con^2 is given by two rules: (1) $Con^1 \subseteq Con^2$, and (2) for any finite nonempty indexing set I, if $c_i \in S$ for all $i \in I$ then $\mathfrak{a}_A(\bigsqcup_{i \in I} c) \in Con^2$. The ordering $\sqsubseteq_2$ is given by the following rules: (1) $\sqsubseteq_1 \subseteq \sqsubseteq_2$, (2) $d \sqsubseteq_2 \mathfrak{a}_A(\bigsqcup_{i \in I} c_i)$ if $d \in$*

Con^1 *and* $d \sqsubseteq_1 \mathfrak{g}_A(\bigsqcup_{i\in I} c_i)$*, and (3)* $\mathfrak{a}_A(\bigsqcup_{i\in I} c_i) \sqsubseteq_2 \mathfrak{a}_A(\bigsqcup_{j\in J} c_j)$ *if* $\mathfrak{g}_A(\bigsqcup_{i\in I} c_i) \sqsubseteq_1 \mathfrak{g}_A(\bigsqcup_{j\in J} c_j)$*. Furthermore, for all* $i \in A$*, for any* $\mathfrak{a}_A(d) \in Con^2$*,* $\mathfrak{s}_i^2(\mathfrak{a}_A(d)) = \mathfrak{a}_A(d)$ *and for each* $e \in Con^1$*,* $\mathfrak{s}_i^2(e) = \mathfrak{s}_i^1(e)$*.*

The next theorem states the correctness of the above construction. Intuitively, the lattice $\mathbf{C^2}(S)$ above must be an scs and the announcement of a certain fact in $c \in S$ must behave similarly to common knowledge or global information of the same fact.

Theorem 6. *Let* $\mathbf{C^1} = (Con^1, Con_0^1, \sqsubseteq_1, ...)$ *be a continuous space-compact* n*-scs (*n*-ecs) and let* $S \subseteq_{fin} Con_0^1$*. Let* $\mathbf{C^2}(S) = (Con^2, Con_0^2, \sqsubseteq_2, ...)$ *as in Def. 12, then (1)* $\mathbf{C^2}(S)$ *is a continuous, space-compact* n*-scs (*n*-ecs), (2)* $\forall \mathfrak{a}_A(c) \in Con^2$*,* $\mathfrak{a}_A(c) \in Con_0^2$*, and (3)* $\forall d \in Con^1$*,* $\forall \mathfrak{a}_A(c) \in Con^2$*,* $d \sqsubseteq_2 \mathfrak{a}_A(c)$ *iff* $d \sqsubseteq_1 \mathfrak{g}_{\mathcal{A}}(c)$*.*

Related Work. There is a huge volume of work on epistemic logic and its applications to distributed systems; [11] gives a good summary of the subject. This work is all aimed at analyzing distributed protocols using epistemic logic as a reasoning tool. While it has been very influential in setting the stage for the present work it is not closely connected to the present proposal to put epistemic concepts into the programming formalism.

Epistemic logic for process calculi has been discussed in [7, 9, 14]. In all of these works, however, the epistemic logic is defined outside of the process calculus, with the processes as models for the logic, whereas our processes have epistemic (or spatial) logic terms within the constraint system, as well as knowledge or space constructions on the processes.

The issue of extending ccp to provide for distributed information has been previously addressed in [23]. In [23] processes can send constraints using communication channels much like in the π-calculus. This induces a distribution of information among the processes in the system. This extension, however, is not conservative wrt to ccp and hence does not share the goal of the present paper.

Another closely related work is the Ambient calculus [6], an important calculus for spatial mobility. Ambient allows the specification of processes that can move in and out within their spatial hierarchy. It does not, however, address posting and querying epistemic information within a spatial distribution of processes. Adding Ambient-like mobility to our calculi is a natural research direction.

One very interesting approach related to ours in spirit – but not in conception or details – is the spatial logic of Caires and Cardelli [4, 5]. In this work they also take spatial location as the fundamental concept and develop modalities that reflect locativity. Rather than using modal logic, they use the name quantifier which has been actively studied in the theory of freshness of names in programming languages. Their language is better adapted to the calculi for mobility where names play a fundamental role. In effect, the concept of freshness of a name is exploited to control the flow of information. It would be interesting to see how a name quantified scs would look and to study the relationship with the Caires-Cardelli framework.

Finally, the process calculi in [2, 3, 10] provide for the use of assertions within π-like processes. They are not concerned with spatial distribution of information and knowledge. These frameworks are very generic and offer several reasoning techniques. Therefore, it would be interesting to see how the ideas here developed can be adapted to them.

Acknowledgments. We thank Raluca Diaconu for her insights and discussions on some preliminary ideas of this work. This work has been partially supported by the project ANR-09-BLAN-0169-01 PANDA and by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS).

References

1. S. Abramsky and A. Jung. Domain theory. In T. S. E. M. S. Abramsky, D. M. Gabbay, editor, *Handbook of Logic in Computer Science, vol. III*. Oxford University Press, 1994.
2. J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *LICS*, 2009.
3. M. G. Buscemi and U. Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In *ESOP*, pages 18–32, 2007.
4. L. Caires and L. Cardelli. A spatial logic for concurrency - i. *Inf. and Comp.*, 2003.
5. L. Caires and L. Cardelli. A spatial logic for concurrency - ii. *Theor. Comp. Sci.*, 2004.
6. L. Cardelli and A. D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.
7. R. Chadha, S. Delaune, and S. Kremer. Epistemic logic for the applied pi calculus. In *FMOODS/FORTE*, pages 182–197, 2009.
8. F. S. de Boer, A. D. Pierro, and C. Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theor. Comput. Sci.*, 151(1):37–78, 1995.
9. F. Dechesne, M. R. Mousavi, and S. Orzan. Operational and epistemic approaches to protocol analysis: Bridging the gap. In *LPAR*, pages 226–241, 2007.
10. F. Fages, P. Ruet, and S. Soliman. Linear concurrent constraint programming: Operational and phase semantics. *Inf. Comput.*, 165(1):14–41, 2001.
11. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.
12. M. Falaschi, M. Gabbrielli, K. Marriott, and C. Palamidessi. Confluence in concurrent constraint programming. *Theor. Comput. Sci.*, 183(2):281–315, 1997.
13. J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. In *Proc. of Principles of Distributed Computing*, pages 50–61, 1984.
14. D. Hughes and V. Shmatikov. Information hiding, anonymity and privacy: a modular approach. *Journal of Computer Security*, 12(1):3–36, 2004.
15. S. Kripke. Semantical analysis of modal logic. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, 1963.
16. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.
17. J. C. C. McKinsey and A. Tarski. The algebra of topology. *The Annals of Mathematics, second series*, 1944.
18. N. P. Mendler, P. Panangaden, P. J. Scott, and R. A. G. Seely. A logical view of concurrent constraint programming. *Nordic Journal of Computing*, 2:182–221, 1995.
19. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes i and ii. *Information and Computation*, 100:1–77, 1992.
20. P. Panangaden. Knowledge and information in probabilistic systems. In *CONCUR 2008*, volume 5201 of *LNCS*, page 4. Springer-Verlag, 2008. Abstract of invited talk.
21. P. Panangaden, V. Saraswat, P. Scott, and R. Seely. A hyperdoctrinal view of concurrent constraint programming. In *Semantics: Foundations and Applications*, LNCS, 1993.
22. S. Popkorn. *First Steps in Modal Logic*. Cambridge University Press, 1994.
23. J.-H. Réty. Distributed concurrent constraint programming. *Fundam. Inform.*, 1998.
24. V. A. Saraswat. *Concurrent Constraint Programming Languages*. PhD thesis, CMU, 1989.
25. V. A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *POPL'91*, 1991.